Homework Batch I: Matrix Multiplication

Gonzato Andrea

March 18, 2021

All the code of my work related to this homework can be found in this file matrix.py [1].

Exercise 1

The implementation of the function *strassen_matrix_mult()* can be found in the file matrix.py [1]

Exercise 2

The implementation of the general case can be found in the function *matrix_mult_generalization()*. Which can perform the multiplication of two matrices of any size $(n \times p)(p \times m)$.

The idea for this function is to up scale the dimensions of the couple of matrices to the closest power of 2. To fills each of those bigger matrices we put the original matrix in the top left corner and fills the rest of the free space whit zeros. Then with these new matrices we can compute the Strassen algorithm and at the end we just need to down scale the matrix obtain after the multiplication.

Follow the demonstration of the time complexity of the algorithm:
Let $g(n, m, p) := next\_power\_of\_2(max(n, m, p))$

- IF (n > m and n > p)
  Let $f(n) := next\_power\_of\_2(n)$
  $g(n, m, p) = f(n) \in \Theta(n)$
  The time complexity $T(n, m, p)$ of the algorithm can be express as:
  $T(n, m, p) = 2 \cdot \Theta(f(n)^2) + T_s\left(f(n)^{\log_2 7}\right)$
  Where the first term is due to the creation of the two bigger matrices and the second term is the time complexity of the execution of the Strassen algorithm.
  $T(n, m, p) = \Theta(n^2) + \Theta\left(n^{\log_2 7}\right) = \Theta\left(n^{\log_2 7}\right)$
  An analog demonstration can be given for the other two cases
- IF (m > n and m > p)
  ...
- IF (p > n and p > m)
  ...

In conclusion we can say that the time complexity of this algorithm is: $\Theta\left(k^{\log_2 7}\right)$
Where $k := max(n, m, p)$

Exercise 3

To improve the implementation of the previous function and reduce the number of auxiliary matrices. I decided to make the algorithm faster by using less operation and less matrices when is try to multiply two matrices and one of those is a null matrix which elements are all zeros.

To make the improvement I changed the *strassen_matrix_mult()* and *matrix_mult_generalization()* functions by adding an optional boolean parameter (*check_null_matrix*). When this parameter is true before run the original Strassen algorithm I check if one of the two matrices is null. If this is the case the multiplication is trivial and I do not need to use the Strassen algoritm.

To evaluate the effect of this improvement I did a test. I generated two random matrices of size $(n \times p)(p \times m)$ and evaluated the time, in seconds, for the multiplication using the standard algorithm and the improved one. Below there is a table of the results of the test obtained using my computer.

n	p	m	standard	improved
3	5	10	0.002	0.002
5	9	18	0.013	0.007
9	17	34	0.085	0.025
17	33	66	0.504	0.142
33	65	130	3.450	0.748
65	129	258	23.926	4.164
129	257	514	169.922	24.616

Exercise 4

References

[1] Andrea Gonzato: Python implementation of matrix multiplication,
`https://github.com/AndreaGonzato/Stassen_Algorithmic_Design/blob/main/matrix.py`

[2] Gary Miller: Introduction and Strassen's Algorithm, Section 2.5 Running Strassen's Algorithm using only $O(n^2)$ space
`http://www.cs.cmu.edu/afs/cs/academic/class/15750-s17/ScribeNotes/lecture1.pdf`